

RABT: Run-Ahead Block TAGE

Prakhar Gupta, Yuwei Sun, Rishav Sen, Reet Sinha, Swetha Karthikeyan
University of Illinois Urbana-Champaign
Champaign, IL, USA
visitprakhar@gmail.com, {yuweis2, rishavs3, reet2, swethak2}@illinois.edu

Abstract—CBP-NG’s VFS metric for branch predictors makes latency, throughput, and energy first-class design constraints alongside accuracy. This paper presents Run-Ahead Block TAGE (RABT), a 75 KB block-level TAGE predictor that issues ahead-of-use table reads and predicts up to seven conditional branches in a 256-instruction fetch block. To make run-ahead prediction robust to successor ambiguity, RABT uses a compact secondary tag derived from the successor block’s PC. On the CBP-NG suite, RABT achieves a VFS of 0.897, with an MPKI of 8.96, IPC of 9.01, and EPI of 2837 fJ, while reducing prediction latency to under one cycle. We characterize limitations of this predictor to outline future directions for ahead-block predictor work.

I. INTRODUCTION

CBP-NG’s VFS-based scoring evaluates branch predictors by accuracy, latency, throughput, and energy. This shifts the design objective away from maximizing accuracy alone and toward predictors that can deliver low-latency, high-throughput predictions within a tight energy budget.

Prior CBP predictors, especially TAGE-SC-L variants, achieve high accuracy with large storage budgets and long resolution chains [1]. Under VFS-based scoring, increased predictor complexity is penalized with increased latency and energy. Figure 1 shows that, across a range of MPKI operating points, the VFS gain from a one-cycle latency reduction consistently exceeds the marginal value of further accuracy improvement, motivating a latency-first design. Run-ahead prediction is attractive in this setting because it moves table accesses off the timing-critical prediction path and eliminates short mispredictions by avoiding multi-layer predictor disagreements.

This paper presents Run-Ahead Block TAGE (RABT), a predictor that combines TAGE [3], [8] with the run-ahead prediction approach of Cai et al. [2]. RABT extends run-ahead TAGE prediction to fetch-block granularity, predicting up to $N=7$ conditional branches in a 256-instruction block at sub-cycle latency. We make the following contributions. We extend run-ahead TAGE prediction to large fetch blocks, amortizing SRAM activation energy across more instructions in low branch-density programs. We floorplan RAMs to reduce critical-path wire delay and meet sub-cycle latency. We use a μ architectural monitor to characterize the accuracy and energy limitations of block-level ahead prediction, identifying targets for future ahead-block predictor designs.

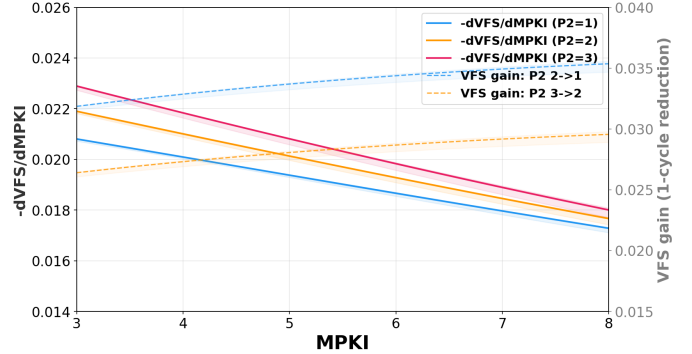


Fig. 1. Marginal VFS Value: Accuracy vs. Latency Reduction

II. DESIGN OVERVIEW

A. Ahead Pipeline

Figure 2 shows RABT’s two-stage ahead pipeline. The predictor operates on 256-instruction fetch blocks containing up to $N=7$ conditional branches. Unlike prior run-ahead TAGE work [2], which runs a fixed number of branches ahead, RABT runs one fetch block ahead and produces an N -wide prediction vector each cycle.

Table reads for a fetch block are performed one cycle before the corresponding predictions are issued. When RABT issues predictions for block B , the data used for that prediction was read from SRAM in the previous cycle, when RABT was predicting block $B - 1$. In the same cycle that RABT predicts B , it initiates the ahead reads needed for predictions in the next cycle. RABT moves SRAM access and primary-tag comparison into the *prefetch* stage. The *predict* stage then performs secondary-tag validation, provider selection, and alternate selection. This split removes SRAM access from the timing-critical prediction path, enabling sub-cycle prediction latency.

Block-level run-ahead prediction introduces a complication. A block with N conditional branches has N possible taken targets plus one fall-through, giving $N+1$ distinct candidate next blocks (*successors*), of which only one will be the true next block. Speculatively reading tables for all successors is impractical because it would require multi-porting tables, incurring considerable area, power and delay costs. Instead, RABT performs a single lookup indexed by the current block’s PC, independent of which successor is eventually reached. Each entry stores a compact hash of the successor PC it is

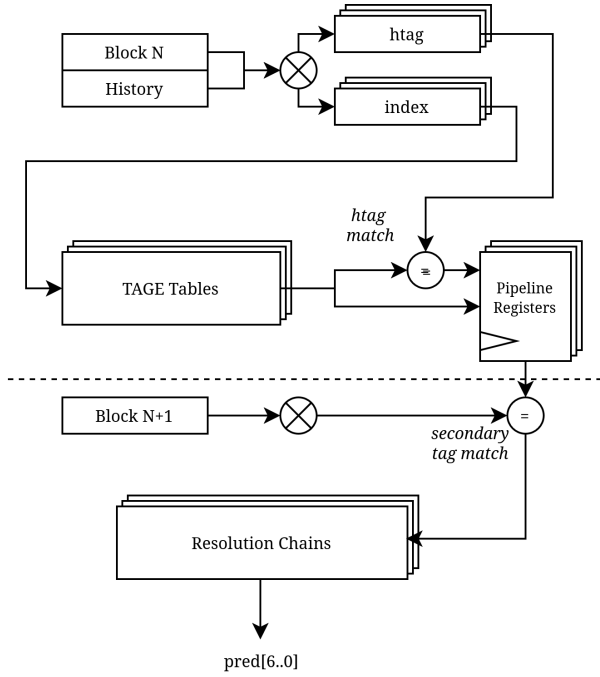


Fig. 2. Two-Stage Ahead Pipeline

intended to serve. At predict time, RABT recomputes the secondary tag from the current block's PC and rejects entries whose stored tag does not match at the cost of different successors of the same block competing for entries at the same index.

Training for a block occurs in the same cycle as its predict phase. On a misprediction, training consumes an additional cycle during which prefetch and predictions for other blocks are suspended. Figure 3 shows the pipelining of the prefetch, predict, and train stages, including an instance of a two-cycle training event on a misprediction.

a) Prefetch stage: This stage issues SRAM reads for the next block across all 15 TAGE tables. For each table, RABT forms the index by XORing the folded global history register with $PC \gg 2$. Each read returns the primary tag, secondary tag, prediction counter, hysteresis, and usefulness values (Figure 4). The primary tag is split into an 8-bit hashed tag (htag) and a 3-bit group ID encoding the branch slot as shown in Figure 6. The htag hit is computed immediately and, together with the secondary tag, prediction, hysteresis, usefulness, and group ID, latched into pipeline registers for the predict stage.

b) Predict stage: The predict stage resolves predictions for the current block using the registers filled during prefetch. RABT computes a 5-bit secondary tag from the current block's PC and rejects entries whose stored tag differs. For each of the N branch slots, RABT then runs an independent TAGE resolution chain. The longest matching table supplies the provider, the second-longest supplies the alternate. If the provider is weak and the meta counter favors the alternate, RABT uses the alternate prediction; otherwise, it uses the provider prediction.

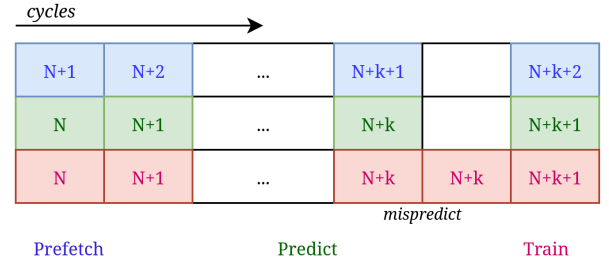


Fig. 3. Pipeline Operation Across Cycles

c) Training stage: Training operates on the same pipeline-register state used during prediction. On each resolved block, the provider's hysteresis counter is updated (increment on correct, decrement on mispredict), and its usefulness counter is incremented when the provider was correct and the alternate disagreed. The meta counter is updated when the provider was weak and provider/alt disagreed. On misprediction, allocation searches tables above the provider for an entry with $u=0$, allocates a single entry with the actual outcome and zeroed counters, and on failure decrements usefulness counters above the provider. Mispredicted blocks require an additional cycle: the fallback hysteresis must be read before the fallback update can proceed, and allocation must write to the single-ported tag and secondary-tag RAMs without conflicting with the ahead reads.

B. Table Configuration

Figure 4 summarizes the final TAGE table configuration; the full parameter list and capacity breakdown are in the appendix (Tables II and III). The total RAM budget is 75 KB across 15 tables. History lengths follow a geometric series from 8 to 200; a 16th table broke timing (+27% P2 latency) without improving MPKI. Table sizes use a two-tier scheme: 1024 entries for the longest-history tables (T0–T4) and 2048 entries for the shorter-history tables (T5–T14), reflecting that long-history tables have lower tag-hit rates and benefit less from extra capacity.

a) Entry-point indexing: Table indices XOR folded global history [5] with $PC \gg 2$ (word-aligned), making each index unique to the entry point at which a fetch block was reached. Tags use $PC \gg 5$, coarser than the index, so the tag identifies a region while the index distinguishes paths into it. This decouples region identity from entry point and lets the predictor maintain distinct history-trained entries for different paths reaching the same code.

b) Floorplan-aware RAM ordering: The physical placement of predictor RAMs has a direct effect on critical-path wire delay and on bit-line switching energy. Each TAGE table's RAMs are organized into two clusters separated by a zone boundary. The *predict cluster* (tag, fold, sec) contains the RAMs read on the timing-critical path during P1; placing the folded-history register immediately after the tag RAM minimizes wire delay on the tag comparison, which is the tightest path for sub-cycle latency. The *update cluster* (pred,

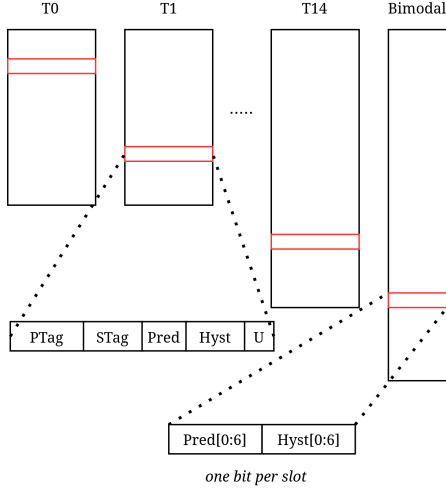


Fig. 4. RABT Tables

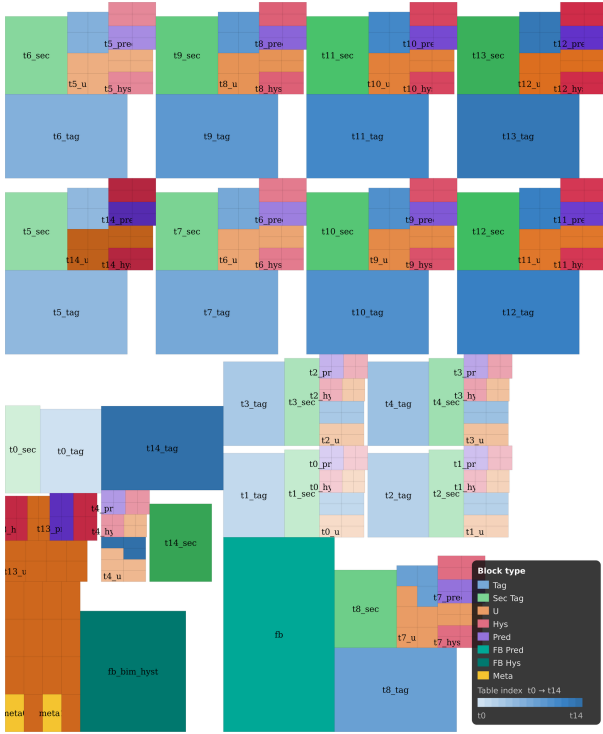


Fig. 5. Physical RAM Layout

hyst, u) contains the remaining RAMs, which tolerate longer wires because their reads are not on the prediction path. Figure 5 shows the resulting layout.

C. Independent Resolution Chains

With $N=7$ branch slots per block, we considered two ways to share TAGE state across slots. *Packed* resolution stores all N predictions in a single entry per index, so every branch in a block is served at the same history depth. *Independent* resolution gives each slot its own tag, hysteresis, and usefulness fields, allowing different slots in the same block to select

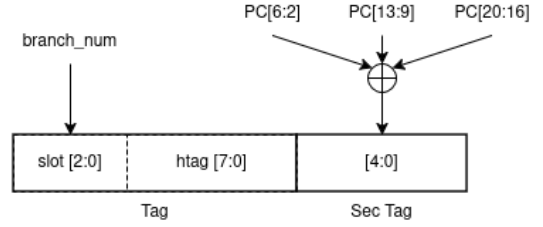


Fig. 6. Tag Format

different provider and alternate tables. Independent resolution preserves the per-branch history-length selection that motivates TAGE in the first place, yielding a 2.0% MPKI improvement and 7.2% EPI reduction over packed resolution on average across the 168 traces. It regresses on workloads with strong intra-block correlation (Appendix B).

III. DETAILED OPERATION

A. Secondary Tag

The secondary tag is a 5-bit hash of the successor block's PC, formed by XORing three non-overlapping PC bit slices (Figure 6). Block PC suffices to identify which of the $N+1$ candidate successors an entry was allocated for, so we use it directly rather than tracking missing history bits as in [2]. The appendix reports the full width and hash sweep.

B. Banked RWRAM

Our ahead design issues reads every cycle for the next block while training writes arrive from resolved blocks, potentially targeting the same bank. Each table is split into 8 banks [9], indexed by a uniform bit-shift of the table address. Banking avoids most read-write conflicts and reduces the SRAM area activated per write. On a conflict the write is buffered and retried on the next cycle; if a second write arrives before the buffer drains, the older one is lost. 8 banks with uniform indexing was optimal; fewer increase write loss, more add latency. The appendix reports the bank sweep and write-loss characterization.

C. Pressure Counters

Two saturating 10-bit counters track predictor metrics. The *accuracy counter* increments on correct predictions and decrements on mispredictions. The *allocation counter* increments when no allocation slot is found and decrements when allocation succeeds. We tested a number of dynamic policies driven by these counters (gated decay, adaptive thresholds, secondary tag enforcement). Only allocation skip proved beneficial, described in the next subsection.

D. Allocation and Decay

On a misprediction, RABT allocates one entry above the provider; probabilistic multi-entry allocation was tested but polluted the tables (-3.3% VFS). During allocation, entries whose htag matches but secondary tag or group ID differs are skipped, promoting allocation to the next table; this avoids

evicting useful entries from other successor paths or branch groups [2]. When the allocation counter indicates high pressure, the closest candidate table is probabilistically skipped, pushing allocation toward longer-history tables. When no table has a useless entry ($u=0$), allocation is suppressed entirely.

Usefulness is managed via LFSR-gated probabilistic decay [7]. On each primary or secondary tag miss, an 8-bit LFSR is compared against a threshold; with the chosen threshold, the matched entry’s u -bit is decremented on approximately 3.1% of decay checks. Entries being allocated in the same cycle are exempt. Epoch-based bulk reset, graded LFSR widths, and per-table thresholds were tested but uniform probabilistic decay proved more stable (see appendix).

E. Fallback Predictor

The fallback is a direct-mapped bimodal table with 8192 entries, indexed by $PC \gg 2$; this preserves entry-point disambiguation within a block ($PC \gg 5$ collapses them and loses 0.8 MPKI). The fallback participates as the lowest-priority “table” in provider selection, with its hit bit hardwired to 1. A separate half-sized RAM stores per-branch hysteresis with a “flip only when wrong twice” policy, reducing fallback MPKI by 9.1% and improving VFS by 1.9%.

F. Meta Corrections

Following [3], we use a pipelined meta corrector to apply bias corrections to alternate/provider selection [4]. When the provider is weak and the provider and alternate disagree, a 4-bit signed counter (2048 entries, pipeline depth 2) determines whether to trust the provider or fall back to the alternate.

G. Other Optimizations

All SRAM writes include a value-change check [3] to eliminate redundant writes for power efficiency. Prediction counters flip only when hysteresis is weak and the prediction was wrong, further reducing write activity.

IV. METHODOLOGY

Development follows a two-step process: (1) a heavily-parameterized HARCOT [10] behavioral model with a μ architectural monitor, used for rapid parameter sweeps, and (2) a hand-coded HARCOT model with explicit floorplanning for accurate area and timing. We iterate between the two, using gradient ascent on VFS to converge on the final configuration. Evaluation used three tiers: a quick evaluation (20 traces), full evaluation (all 168 CBP-NG traces), and μ Arch monitor evaluations (deep single-trace instrumentation for diagnostics). See the appendix for details.

V. RESULTS

a) Accuracy: RABT increases MPKI by 3.47 relative to the example 2-cycle TAGE across the 168 released traces (Figure 7). For comparison, Cai et al. [2] report only +0.1 MPKI for their direct-vs. ahead TAGE comparison.

Although the 256-instruction fetch block appears to be an obvious source of accuracy loss, shrinking the block width reduced MPKI by only 1.1%. Thus, block granularity explains

little of the gap. A larger contributor is the fallback predictor. RABT uses a block-level, 7-wide bimodal fallback, which underperforms the conventional bimodal fallback used in the example TAGE by 1.376 MPKI. Because approximately 65% of RABT predictions are supplied by the bimodal component, we attribute roughly 0.894 MPKI of the accuracy loss to this fallback difference.

A second contributor is short-history coverage. RABT’s minimum tagged history length is 8, whereas the example TAGE begins at history length 2. Changing RABT’s history series to start at 2 reduces MPKI by 0.923 across all traces. Together these two choices explain 1.817 MPKI, or 52% of the gap. The remaining 48% concentrates in eight traces that regress by more than 10 MPKI against the example TAGE and requires more work; the μ Arch counters for those traces appear in Table VIII.

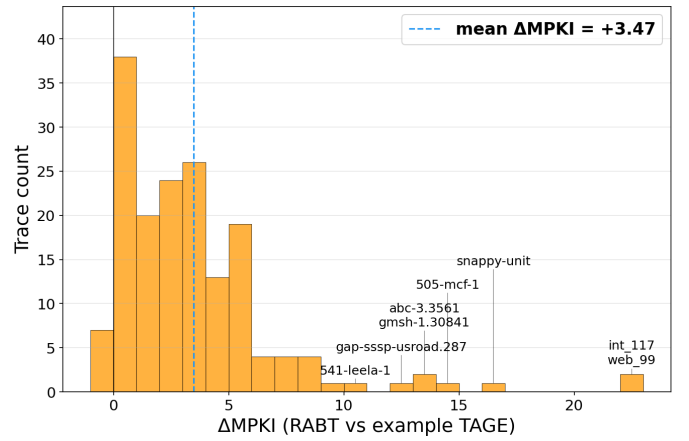


Fig. 7. Per-Trace Δ MPKI: example 2-Cycle TAGE vs. RABT

b) Power: RABT consumes $2.23\times$ more energy per correct-path instruction than the example TAGE predictor. This increase is only partly explained by capacity: RABT has $2.14\times$ the storage and $2.20\times$ the static power of the example TAGE. Dynamic power grows more sharply, by $3.36\times$. Table I shows that the excess dynamic energy is dominated by wiring. Wiring energy grows by $3.73\times$ and accounts for over 90% of the total EPI gap, while RAM+logic energy grows by only $1.19\times$.

To separate storage and table-count effects from RABT-specific overheads, we compare against a *scaled TAGE* configuration: the example TAGE grown to RABT’s 15 tables and comparable storage, but omitting the secondary-tag arrays. We parse the HARCOT floorplan files for each predictor and tally the number of wire bundles and the sum of their Manhattan lengths; Table VII in the appendix reports these statistics.

Compared with scaled TAGE, RABT has $1.91\times$ more wire bundles and $3.04\times$ greater total wire length. This additional wiring comes from RABT’s 8-way RAM banking and per-table secondary-tag arrays, which increase the number of physical SRAM instances and the address/select wiring needed to access them. The wire distribution appears in Figure 10 in the appendix. RABT’s high EPI is thus primarily a wiring cost.

TABLE I
POWER / STORAGE BREAKDOWN

	example TAGE	RABT	Increase
<i>EPI (fJ)</i>			
RAM + Logic	716	850	1.19×
Fanout	29	26	0.90×
Wiring	526	1961	3.73×
Total	1271	2837	2.23×
<i>HARCOM panel metrics</i>			
Storage (Kbit)	281	603	2.14×
Transistors (M)	2.01	4.26	2.12×
SRAM area (mm ²)	0.0132	0.0265	2.01×
<i>Power (mW)</i>			
Dynamic	25.6	85.9	3.36×
Static	0.41	0.90	2.20×

Cai et al. [2] report a $+1.5\times$ energy increase for their ahead predictor, but measure only TAGE-table SRAM activation. RABT's RAM-side energy only grows by $1.19\times$. RABT also consumes 21% less RAM+logic energy than the scaled TAGE configuration. This reflects the advantage of the 256-instruction fetch block, which amortizes SRAM activation over more correct-path instructions, especially in low-branch-density regions. Figure 8 shows this trend across the most power-impacted traces.

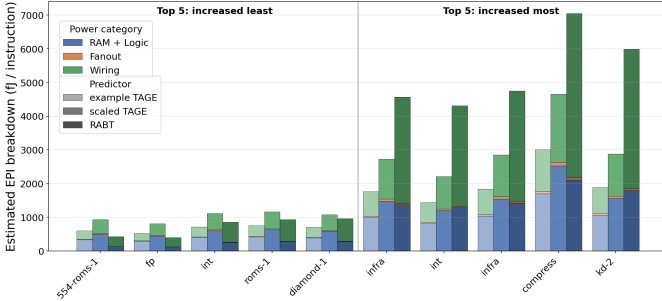


Fig. 8. Per trace power distribution

c) VFS Impact: Despite the accuracy penalty, the VFS gain from latency reduction (2 cycles $\rightarrow$ 1 cycle) more than compensates under the CBP-NG scoring framework. Table IV (appendix) decomposes the VFS improvement incrementally through the design space exploration for RABT. Additional per-trace results, design space exploration (banking, history lengths, tag widths), and parameter sweep data are in the appendix.

VI. DISCUSSION

a) VFS landscape: Figure 9 plots VFS against MPKI for the predictors shipped with the CBP-NG framework (bimodal, gshare, TAGE, and their banked variants) alongside RABT. As a high-accuracy reference, we additionally plot an estimated 64 KB TAGE-SC-L [9] using the CBP-5 reference MPKI of 3.50, the example TAGE's IPC and throughput, and an EPI estimated as the example TAGE's EPI (1246 fJ) plus

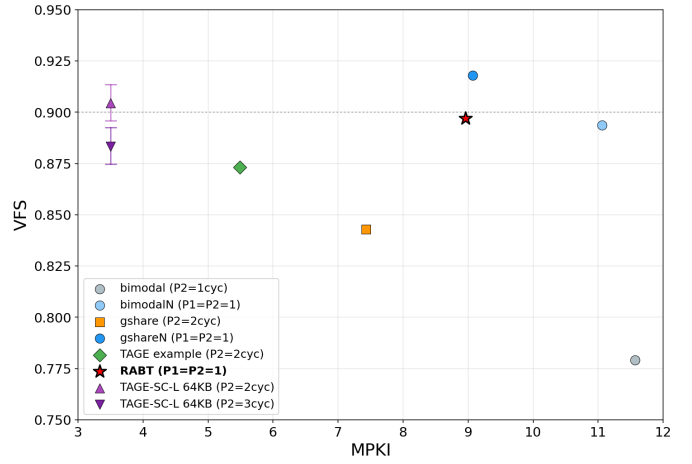


Fig. 9. VFS vs. MPKI for Different Predictors

1800–3000 fJ for 6–8 additional SC table reads. Two latency scenarios are shown: P2=2 cycles (optimistic, assuming SC is pipelined away or omitted) and P2=3 cycles (the typical case in which SC index computation serializes against the TAGE provider). TAGE-SC-L leads RABT on VFS in both scenarios, but the margin is smaller than its $2.5\times$ MPKI advantage alone would suggest, and narrows further at P2=3. The TAGE-SC-L estimate carries wide uncertainty: the EPI whiskers reflect the 1800–3000 fJ SC-overhead range, and even the most favorable point in that range falls short of the gap that the MPKI advantage implies.

b) Run-ahead overrides: RABT does not include the statistical correlator (SC), loop predictor (L), or IMLI [6] components that boost TAGE-SC-L accuracy. Porting SC specifically is non-trivial in a run-ahead architecture: its inputs are not available until the TAGE resolution chain completes, which is the same cycle in which the prediction must be delivered.

c) Future work: With prediction latency reduced to one cycle, future gains should target RABT's remaining MPKI and wiring-energy bottlenecks. The accuracy results suggest that stronger short-history coverage and a better block-level fall-back predictor can recover a substantial fraction of the gap to the example TAGE. Statistical correlator architectures adapted for ahead operation could recover much of the accuracy gap to TAGE-SC-L. BTB integration with shared entries could extend the ahead block distance. A prediction dispatch queue is another promising avenue.

VII. CONCLUSION

We have presented Run-Ahead Block TAGE (RABT), a branch predictor that adapts ahead pipelining to block-level TAGE with independent resolution chains, secondary-tag disambiguation, and floorplanned RAMs. On the CBP-NG suite, RABT achieves a VFS of 0.897, with an MPKI of 8.96, IPC of 9.01, and EPI of 2837 fJ. Under VFS-based scoring, RABT's one-cycle prediction latency offsets its accuracy and energy penalties. By addressing these limitations and iterating further

in the design space, we believe run-ahead block prediction offers a promising avenue for high-performance branch prediction.

REFERENCES

- [1] A. Seznec, “TAGE-SC for CBP2025,” presented at CBP, Tokyo, Japan, Jun. 2025.
- [2] L. Cai, A. Deshmukh, and Y. Patt, “Enabling ahead prediction with practical energy constraints,” in *Proc. ISCA*, 2025, doi: 10.1145/3695053.3730998.
- [3] A. Seznec, “A new case for the TAGE branch predictor,” in *Proc. 44th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, Porto Alegre, Brazil, 2011, pp. 117–127.
- [4] A. Seznec, “A 256 Kbits L-TAGE branch predictor,” *J. Instruction-Level Parallelism*, vol. 9, 2007. [Online]. Available: <http://www.jilp.org/vol9>
- [5] A. Seznec, S. Felix, V. Krishnan, and Y. Sazeides, “Design tradeoffs for the Alpha EV8 conditional branch predictor,” in *Proc. 29th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2002, pp. 295–306.
- [6] A. Seznec, J. San Miguel, and J. Albericio, “The inner most loop iteration counter: A new dimension in branch history,” in *Proc. 48th Int. Symp. Microarchit. (MICRO)*, 2015, pp. 347–357.
- [7] A. Perais and A. Seznec, “Cost effective speculation with the omnipredictor,” INRIA, Rennes, France, Tech. Rep., 2015.
- [8] A. Seznec and P. Michaud, “A case for (partially) tagged geometric history length branch prediction,” *J. Instruction-Level Parallelism*, vol. 8, 2006. [Online]. Available: <http://www.jilp.org/vol8>
- [9] A. Seznec, “TAGE-SC-L branch predictors again,” in *Proc. 5th Championship Branch Prediction (CBP-5)*, 2016.
- [10] P. Michaud, “HARCOM,” INRIA, 2024. [Online]. Available: <https://gitlab.inria.fr/pmichaud/harcom>

APPENDIX

TABLE II
TAGE TABLE PARAMETERS

Parameter	Value
Number of tables (NT)	15
History range	$H_{\min}=8$ to $H_{\max}=200$ (geometric)
History lengths	{200, 158, 126, 100, 79, 63, 50, 40, 31, 25, 20, 15, 12, 10, 8}
Path history width	27 bits
Path bits per cycle	6 bits of next-PC
History buffer	8192 entries
Table sizes	1024 (T0–T4), 2048 (T5–T14)
Tag width	Uniform 11 bits (all tables)
Prediction counter	1-bit (CTR_WIDTH=1)
Hysteresis	2-bit (HYST_WIDTH=2, separate [5])
Usefulness counter	2-bit (U_WIDTH=2)
Secondary tag	5-bit (hash of successor PC)
Banks per table	8 (BANK_SHIFT=1)

A. Write loss and bank conflicts

The μ Arch monitor revealed 30–64% write loss on short-history tables (T13–T14) with 8-way banking; long-history tables showed near-ideal bank distribution. Per-bit flip rate instrumentation shows that short-history fold registers shift by only 1 bit per cycle, so consecutive indices differ by very few bits. At most 1 of the 3 bank-select bits flips between consecutive accesses, producing 56–57% same-bank rates on these tables.

We attempted several mitigations. Counter-based bank rotation increased write loss to 78% and MPKI by 97%, because rotating the bank bit changes which physical entry is accessed. XOR-ing block PC into bank select failed because the PC

TABLE III
RAM CAPACITY BREAKDOWN (75 KB TOTAL)

RAM type	Bits	KB	%
Tag (11b)	281,600	34.38	45.8
Sec tag (5b)	128,000	15.62	20.8
Fallback (7b)	57,344	7.00	9.3
Usefulness (2b)	51,200	6.25	8.3
Fallback hyst (7b)	28,672	3.50	4.7
Prediction (1b)	25,600	3.12	4.2
Hysteresis (2b)	25,600	3.12	4.2
Meta (4b)	16,384	2.00	2.7
<i>By table group (%)</i>			
T0–T4 (1024×5)		12.50	
T5–T14 (2048×10)		50.00	
FB + FB_BH		10.50	
Meta		2.00	

TABLE IV
CUMULATIVE ABLATION

Design point	Δ VFS	MPKI	P2 (cyc)	EPI (fJ)
2-cycle TAGE (example)	—	5.49	2	1246
RABT (shared resolution)	+0.9%	9.15	0.890	2571
+ Independent chains	+1.3%	9.09	0.867	2386
+ Fallback hysteresis	+2.7%	8.96	0.913	2837

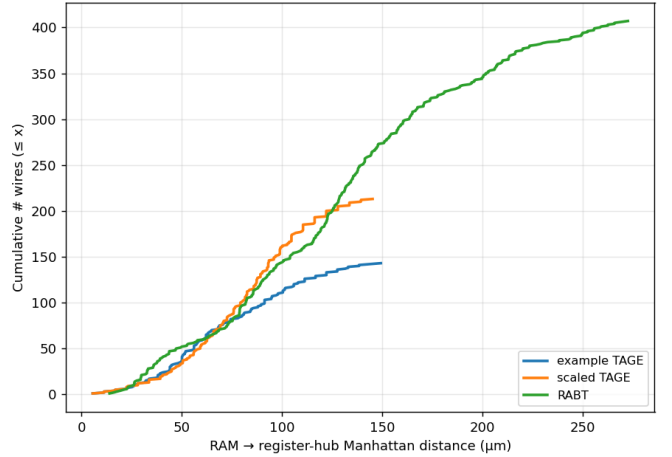


Fig. 10. Wire distribution

is constant within a self-succeeding loop. Per table bank indexing (+0.27 MPKI, 5/20 wins) and uniform bank indexing with offsets = 5 (+0.8 MPKI) were both strictly worse. In hot self-looping blocks, read-write conflicts on short-history tables cannot be mitigated regardless of bank selection scheme. However, if the loop continues without mispredictions, the skipped training writes are benign since no update is needed. As such, this effort was concluded.

B. Workload sensitivity to entry sharing

Independent resolution trades intra-block correlation for per-branch independence. This works well on average (−2.0% MPKI) but regresses on workloads with strong intra-block correlation (e.g. dcapo-kafka +63.9%, sampleflow +35.9%).

TABLE V
PARAMETER SWEEP SUMMARY

Parameter	Final	Alternatives tested
NT	15	14, 16 (timing fail)
$H_{\max}$	200	150, 300
Tag	Uniform 11b	Graded $\langle 13, 9 \rangle$, $\langle 11, 7 \rangle$
SEC_TAG	5-bit hash	3b, 4b, adaptive, press-gated
MAX_ALLOC	1	2 (−3.3% VFS)
FB capacity	8192	16384 (VFS drop from EPI)
META	4-bit/2048	2-bit/1024
Banks	8	2, 4 (write loss), 16 (EPI penalty)
BANK_SHIFT	Uniform 1	Graded $5 \rightarrow 1$ (+0.27 MPKI)
PC_IDX_SHIFT	2	5 (loses PC entropy, +0.8 MPKI)
FB banking	Single $8K \times 7$	4 addr banks (+21% P2)
FB hysteresis	4096×7	None (−1.9% VFS)
Decay	LFSR=8, thresh=8	Graded LFSR/thresh (all worse)

TABLE VI
SECONDARY TAG SWEEP

Bits	Hash ($a \oplus b \oplus c$)	Rej.%	MPKI Δ
0	—	0%	+1.8%
3	PC _{4:2} , PC _{9:7} , PC _{14:12}	22.9%	+3.5%
4	PC _{5:2} , PC _{11:8} , PC _{16:13}	23.4%	+2.4%
5	PC _{6:2} , PC _{13:9} , PC _{20:16}	24.0%	baseline

TABLE VII
WIRELENGTH DIFFERENCES

Predictor	Wires	$\Sigma d_{\text{reg-ram}}$ mm	Mean μm	Median μm
Example TAGE	143	10.36	72.4	68.2
Scaled TAGE	213	17.10	80.3	82.2
RABT	407	51.90	127.5	126.2

TABLE VIII
 μ ARCH METRICS FOR OUTLIERS VS. THE 160 NON-OUTLIER TRACES

Trace	blk%	sib%	aloc%	cNoM	cFB
abc-3	14.0	0.1	0.1	99.0	0.5
int_117	14.9	0.6	0.5	86.8	5.2
snappy	16.6	6.6	7.2	74.0	25.1
web_99	25.3	1.0	1.6	0.8	99.1
mcf	0.0	55.5	45.9	14.7	83.1
gmsh	0.0	76.8	39.2	23.8	72.9
gap	0.0	65.3	54.1	10.9	87.3
leela	0.0	35.8	53.0	8.0	91.1
Median	0.15	5.66	9.36	15.60	81.80
Mean	0.45	9.76	10.71	20.68	75.16

blk% = alloc. attempts blocked by non-zero u-bit; sib% = alloc. refused by sibling policy; aloc% = alloc. success rate; cNoM = block-mispred. cause: TAGE no-meta; cFB = block-mispred. cause: bimodal fallback.